



**Instituto de Informática**

---

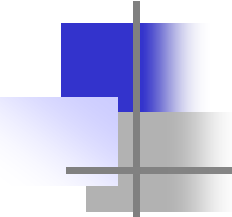
**POO – Turmas B e C**

# **Polimorfismo – Parte 2**

**10/02/2022**

**Prof. Dirson S. Campos**

**Profa. Nádia Félix**



# Polimorfismo e Herança

## (continuação)

---

- ❑ Em POO a herança é usada para o suporte a:
  - reutilização de código, e
  - implementação de polimorfismo.
- ❑ Antes de aplicar o polimorfismo é preciso associar operação, método e invocação (chamada do método no programa principal (main)).



# Polimorfismo e Herança

## (continuação)

---

- ❑ Operação, neste contexto polimórfico, é algo que se pode invocar numa instância de uma classe para atingir determinado objetivo;
- ❑ Método, neste contexto, é a implementação concreta da operação para uma determinada classe;
- ❑ Invocação de uma operação, neste contexto, leva à execução de um método.



# Recordando a ideia de Polimorfismo

---

- ❑ A ideia por trás do polimorfismo é a invocação de uma operação que pode levar à execução de diferentes métodos, desta forma, uma operação polimórfica deve ter no mínimo duas diferentes implementações.
- ❑ O método efetivamente invocado depende da classe do objeto onde a operação é invocada e não depende do tipo da referência utilizada.



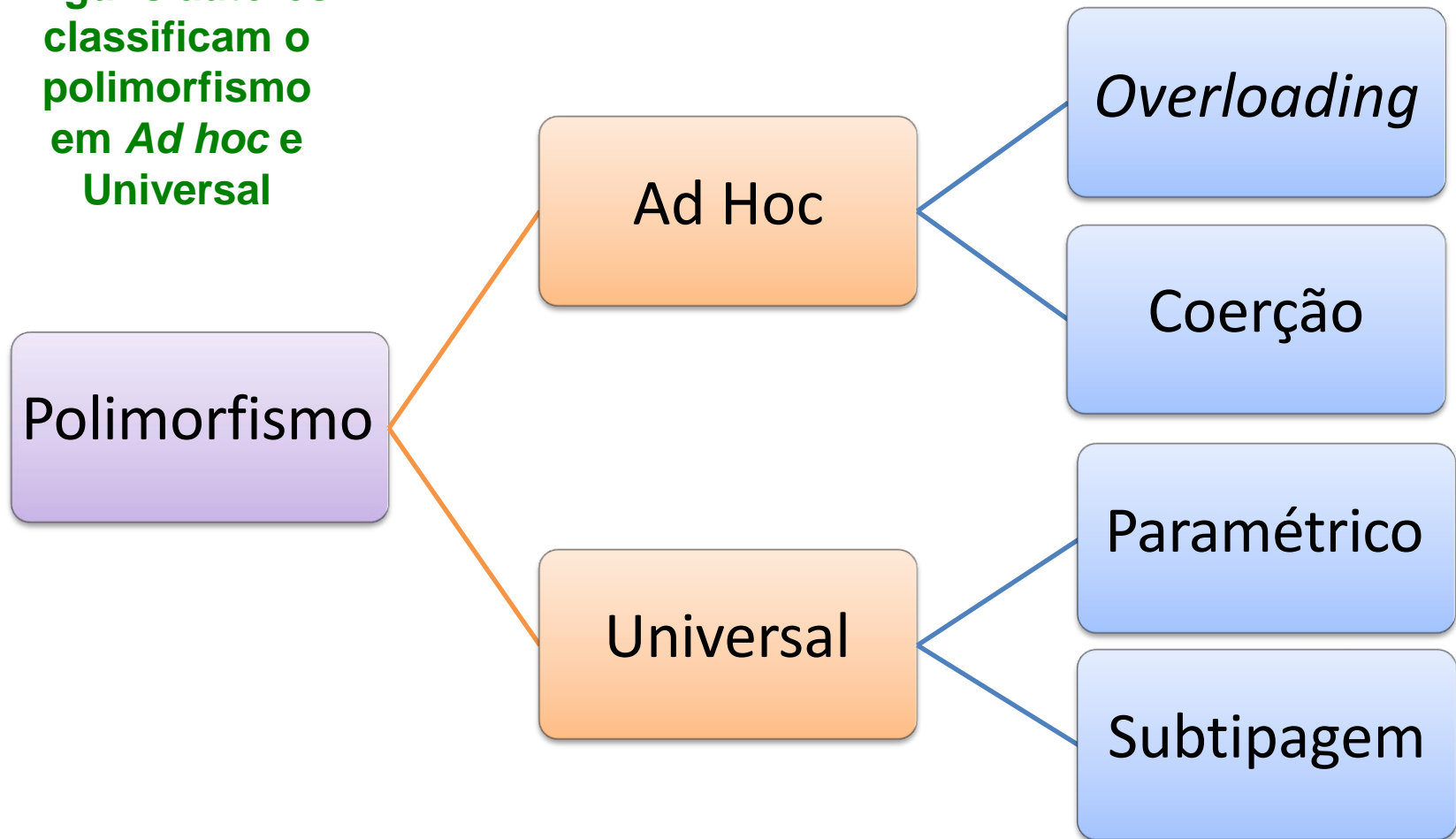
# Recordando a ideia de Polimorfismo

---

- ❑ Um método polimórfico possui várias formas:
  - A “forma” descrita pela classe a que pertence;
  - As “formas” das classes acima na hierarquia a que pertence.

# Taxonomia Polimorfismo

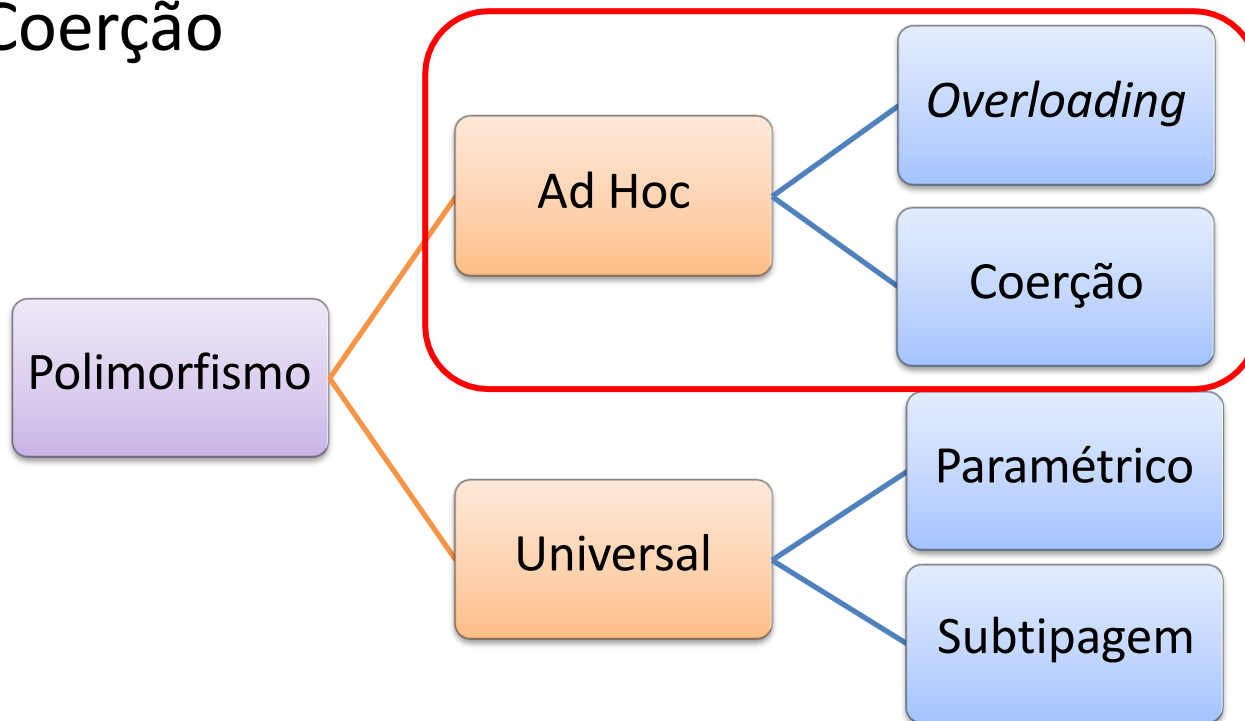
Alguns autores  
classificam o  
polimorfismo  
em *Ad hoc* e  
Universal



# Polimorfismo Ad-hoc

## ❑ Número finito de variações

- Sobrecarga (*overloading*)
- Coerção





# Polimorfismo: Sobrecarga (*overloading*)

---

- ❑ É resolvido estaticamente, em tempo de compilação.
  - O termo sobrecarga (*overloading*) vem do fato de declararmos vários métodos com o mesmo nome, estamos carregando o aplicativo com o 'mesmo' método.
  - A única diferença entre esses métodos são seus parâmetros e/ou tipo de retorno.
    - Logo utiliza-se os tipos para escolher quais dos métodos a serem chamados.





# Exemplo de Polimorfismo (Sobrecarga)

❑ Que função é chamada abaixo para qual método?

```
public class Square {  
  
    public static double square(double a) {  
        System.out.println("Square of double: " + a);    return a *  
        a;  
    }  
  
    public static int square(int a) {    System.out.println("Square  
        of int: " + a);    return a * a;  
    }  
  
    public static void main(String args[]) {  
        square(1);    square(1.0);  
        square('a');  
    }  
}
```

# Verificando as funções do método square no BlueJ

The image shows a screenshot of the BlueJ IDE interface. The main window displays the source code for the `Square` class, which implements a polymorphic `square` method. The code is as follows:

```
public class Square {  
  
    public static double square(double a) {  
        System.out.println("Square of double: " + a);  
        return a * a;  
    }  
  
    public static int square(int a) {  
        System.out.println("Square of int: " + a);  
        return a * a;  
    }  
  
    public static void main(String args[]) {  
        square(1);  
        square(1.0);  
        square('a');  
    }  
}
```

On the left, the 'Projeto' pane shows the class hierarchy with `Square` selected. Below it, the 'Opções' dialog box displays the results of the test execution:

```
Square of int: 1  
Square of double: 1.0  
Square of int: 97
```

An observation is provided in red text:

**Obs.: o caractere 'a' é o número 97 do código decimal (inteiro) da Tabela ASCII**

At the bottom, the 'Executar Testes' button is visible, along with a status bar indicating 'gravando' and a 'Fim' button.



# Outro Exemplo de Polimorfismo de Sobrecarga

❑ E nesse caso qual método sum é chamado?

```
public class TestaSoma {  
  
    public static void soma(int a, int b) {  
        System.out.println("Soma de inteiros: " + (a + b));  
    }  
  
    public static void soma(double a, double b) {  
        System.out.println("Soma de reais: " + (a + b));  
    }  
  
    public static void main(String args[]) {  
        soma(1, 2);  
        soma(1.1, 2.2);  
    }  
}
```

# Verificando as funções do método soma no BlueJ

The screenshot displays the BlueJ IDE interface. The main window, titled 'TestaSoma - Polimorfismo3', shows the source code of the `TestaSoma` class. The code defines two static methods, `soma(int a, int b)` and `soma(double a, double b)`, both of which print the sum of their arguments. The `main` method calls `soma(1, 2)` and `soma(1.1, 2.2)`. To the left, a terminal window titled 'BlueJ: Janela de Terminal - Polimorfismo3' shows the output of the program: 'Soma de inteiros: 3' and 'Soma de reais: 3.3000000000000003'. The top menu bar includes 'Projeto', 'Editar', 'Ferramentas', 'Exibir', and 'Ajuda'. The left sidebar contains buttons for 'Nova Classe...', a right arrow, and a document icon labeled 'TestaSoma'. The bottom right corner of the slide features the number 12.

```
public class TestaSoma {  
  
    public static void soma(int a, int b) {  
        System.out.println("Soma de inteiros: " + (a + b))  
    }  
  
    public static void soma(double a, double b) {  
        System.out.println("Soma de reais: " + (a + b));  
    }  
  
    public static void main(String args[]) {  
        soma(1, 2);  
        soma(1.1, 2.2);  
    }  
}
```

Soma de inteiros: 3  
Soma de reais: 3.3000000000000003

12



# Coerção

---

## □ Coerção

– Foi visto na aula anterior.

- Técnica conhecida também por Casting
- Atribuição forçosa de tipo a objetos recuperados em tempo de execução.
- Utiliza a definição para escolher o tipo de conversão



# Coerção

---

## ☐ Coerção

- Acontece quando um tipo primitivo ou um objeto é ‘convertido’ em outro tipo de objeto ou tipo primitivo, e essas conversões podem ser implícitas (são feitas automaticamente) ou explicitada onde o tipo do casting fica entre parênteses.



# Coerção

---

❑ Trecho de código em Java exemplo de coerção

```
float x = 3.5;
```

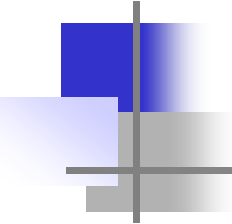
```
int y = 2;
```

```
int soma;
```

```
soma = (int) (x + y); //Resultado 5
```

```
//Coerção explícita com casting
```

```
y = 2.33 //coerção implícita, sem casting
```



# Dependendo do tipo de coerção pode-se modificar o tamanho em bytes utilizados

<b>TIPO</b>	<b>TAMANHO</b>
boolean	1 bit
byte	1 byte
short	2 bytes
char	2 bytes
int	4 bytes
float	4 bytes
long	8 bytes
double	8 bytes





# Exemplo de Polimorfismo de Coerção

❑ E nesse caso qual método sum é chamado?

```
public class TestaSoma1 {  
  
    public static void soma(int a, int b) {  
        System.out.println("Soma de inteiros: " + (a + b));  
    }  
  
    public static void soma(double a, double b) {  
        System.out.println("Soma de reais: " + (a + b));  
    }  
  
    public static void main(String args[]) {  
        soma(5, 2.2);  
        soma((int)5.1, (int)2.2);  
    }  
}
```

# Verificando as funções do método soma no BlueJ

The screenshot displays the BlueJ IDE interface. The main window, titled 'TestaSoma1 - Polimorfismo3', shows the source code of the `TestaSoma1` class. The code defines two static methods, `soma`, for adding integers and doubles, and a `main` method that calls these methods with specific values. The output window, titled 'BlueJ: Janela de Terminal - Polimorfismo3', shows the results of the program execution: 'Soma de reais: 7.2' and 'Soma de inteiros: 7'.

BlueJ:Polimorfismo3

Projeto Editar Ferramentas Exibir Ajuda

Nova Classe...

→

Compilar

TestaSoma1

TestaSoma1 X

Classe Editar Ferramentas Opções

Compilar Desfazer Recortar Copiar Colar Procurar... Fechar Cód

BlueJ: Janela de Terminal - Polimorfismo3

Opções

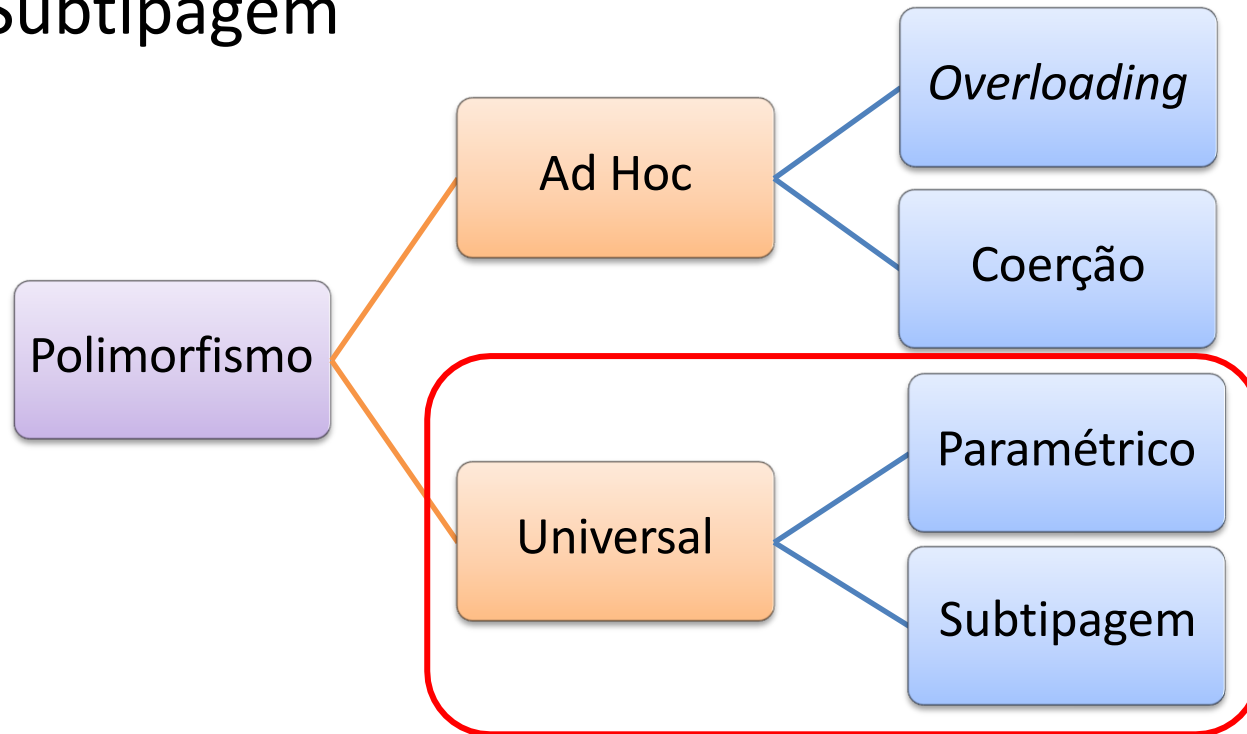
Soma de reais: 7.2  
Soma de inteiros: 7

```
public class TestaSoma1 {  
  
    public static void soma(int a, int b) {  
        System.out.println("Soma de inteiros: " + (a + b));  
    }  
  
    public static void soma(double a, double b) {  
        System.out.println("Soma de reais: " + (a + b));  
    }  
  
    public static void main(String args[]) {  
        soma(5, 2.2);  
        soma((int)5.1, (int)2.2);  
    }  
}
```

# Polimorfismo Universal

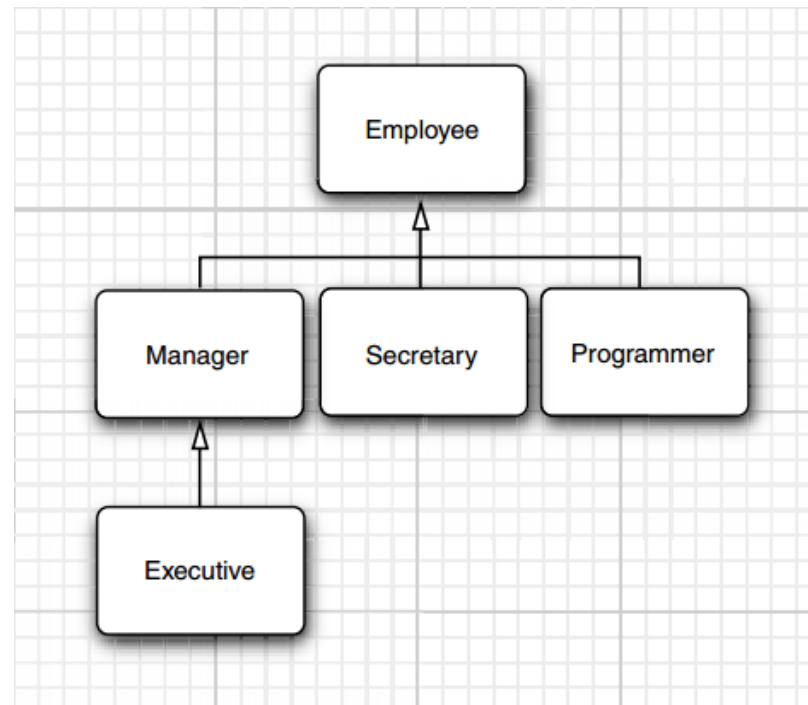
## ❑ Número infinito de variações

- Paramétrico
- Subtipagem



# Polimorfismo Universal

- ❑ O que acontece quando a chamada de um método é feita a objetos de vários tipos de hierarquia?





# Polimorfismo Universal

---

- ❑ O que acontece quando a chamada de um método é feita a objetos de vários tipos de hierarquia? **(Resposta)**
  - A subclasse verifica se ela tem ou não um método com esse nome e com os mesmos parâmetros
  - Senão tiver, a classe que é a uma da(s) superclasse(s) tornasse-a responsável pelo processamento da mensagem.



# Polimorfismo Universal

❑ Ligação tardia (*late binding*) ou ligação dinâmica.

- É a chave para o funcionamento
- O compilador não gera o código em tempo de compilação, ou seja, na ligação tardia ou ligação dinâmica, o compilador não decide o método a ser chamado.
- Desta forma, a definição do método e a chamada do método são vinculadas durante o tempo de execução.
- O objeto real é usado para vinculação dinâmica a execução é mais lenta em comparação a ligação estática.



# Polimorfismo Universal

---

## □ Ligação estática (*early binding*)

- Early Binding: A ligação que pode ser resolvida em tempo de compilação pelo compilador é conhecida como ligação estática ou inicial (*early*).
- A vinculação de todos os métodos estáticos, privados e finais é feita em tempo de compilação.
- O objeto real não é usado para vinculação uma vez que este tipo de objeto é criado em tempo de execução não em tempo de compilação.
- A vinculação estática ocorre na sobrecarga (*overloading*) e é mais rápida do que a ligação tardia.



# Polimorfismo Paramétrico

---

- ❑ É uma forma de se tornar uma linguagem mais expressiva
- ❑ Mantém toda sua tipagem estática segura
- ❑ Foi introduzido em Java 1.5, como uma forma de reuso.
  - Este tipo de polimorfismo é chamado de **generics** *em Java*, ou seja, a implementação Java de polimorfismo paramétrico chama-se **generics**.
  - Em outras linguagens POO pode ter tem outros nomes, em C++ são chamadas de *templates*.





# Polimorfismo Paramétrico

❑ **Generics** em Java

(<https://docs.oracle.com/javase/tutorial/java/generics/why.html>)

❑ Permite aos programadores escreverem **métodos genéricos**

- Os parâmetros dos métodos, variáveis locais e o tipo de retorno podem ser definidos na chamada do método.
- Permite ao mesmo método ser invocado usando-se tipos distintos (sem precisar sobrescrevê-lo)



# Polimorfismo Paramétrico

❑ **Generics** em Java

(<https://docs.oracle.com/javase/tutorial/java/generics/why.html> )

❑ Permite também a definição de **classes genéricas**.

- Os atributos da classe podem ser definidos no momento da instanciação do objeto
- Recurso útil ao definir classes como estruturas de dados



# Polimorfismo Paramétrico

---

- ❑ O polimorfismo paramétrico representa a possibilidade de definir várias funções do mesmo nome mas possuindo parâmetros diferentes (em número e/ou tipo).
- ❑ O polimorfismo paramétrico torna assim possível a escolha automática do bom método a adoptar em função do tipo de dado passado em parâmetro.



# Polimorfismo Paramétrico

---

- ❑ O polimorfismo paramétrico é parecido, mas é diferente do polimorfismo Ad-hoc de sobrecarga (*overloading*), basicamente porque ele é uma função ou um tipo de dado que pode ser escrito genericamente para que seja possível lidar com valores de forma idêntica sem depender do seu tipo de dados.



# Polimorfismo Paramétrico

❑ **Exemplo.** No trecho de código abaixo é criada uma estrutura de dados chamada **Point** do tipo genérico chamado de **T** que tem um tipo **Point** do tipo genérico que foi implementado polimorficamente duas vezes, o primeiro tipo **int** o segundo do tipo **double**.

```
public struct Point<T>
```

```
{  
    public T X;  
    public T Y;  
}
```

```
Point<int> point;
```

```
point.X = 1;
```

```
point.Y = 2;
```

```
Point<double> point;
```

```
point.X = 1.2;
```

```
point.Y = 3.4;
```



# Polimorfismo Subtipagem

- ❑ Na teoria da linguagem de programação , subtipo **(também denominados por alguns autores de polimorfismo de subtipo ou Polimorfismo de Inclusão)** é uma forma de polimorfismo de tipo em que um subtipo é um tipo de dados que está relacionado a outro tipo de dados (o supertipo) por alguma noção de substituíbilidade, por exemplo, numérica.
  - No segundo grau, na Teoria de Conjunto, foi aprendido o operador está contido ( $\subset$ ) , por exemplo,  $\mathbb{Z}$  (nr. inteiro)  $\subset$   $\mathbb{R}$  (nr. real) então um nr.  $\mathbb{Z}$  é um subtipo de nr.  $\mathbb{R}$ , mas nem todo nr.  $\mathbb{R}$  é  $\mathbb{Z}$ .



# Polimorfismo Subtipagem

- ❑ Substituibilidade significa que os elementos do programa, normalmente sub-rotinas ou funções, escritas para operar em elementos do supertipo, também podem operar em elementos do subtipo. Se  $S$  é um subtipo de  $T$ , a relação de subtipagem é frequentemente escrita  $S <: T$ , para significar que qualquer termo do tipo  $S$  pode ser usado com segurança em um contexto onde um termo do tipo  $T$  é esperado.



# Polimorfismo Subtipagem

---

- ❑ A semântica precisa de subtipagem depende crucialmente dos detalhes do que "usado com segurança em um contexto em que" significa em uma determinada linguagem de programação.





# Polimorfismo Subtipagem

---

- ❑ O sistema de tipos de uma linguagem de programação define essencialmente sua própria relação de subtipagem, que pode muito bem ser trivial se a linguagem não oferecer suporte a nenhum (ou poucos) mecanismos de conversão.



# Polimorfismo Subtipagem

---

- ❑ Devido à relação de subtipagem, um termo pode pertencer a mais de um tipo.
  - A subtipagem é, portanto, uma forma de polimorfismo de tipo.



# Polimorfismo Subtipagem

---

- ❑ Na programação orientada a objetos, o termo 'polimorfismo' é comumente usado para se referir apenas a este polimorfismo de subtipo, enquanto as técnicas de polimorfismo paramétrico seriam consideradas programação genérica.
- ❑ O polimorfismo de subtipo aplicado a objetos usa o **Princípio da Substituição formalizado por Barbara Liskov em 1988.**



# Princípio da Substituição aplicado a Objetos (Liskov)

- ❑ Se S é um subtipo de T, então objetos do tipo T podem ser substituídos por objetos do tipo S, sem alterar qualquer propriedade desejável do programa (correção, tarefa executada, etc.)
  - O objetivo ao aplicar este princípio é ter certeza de que novas classes derivadas (subclasses) estão estendendo das classes base (superclasses), mas sem alterar o seu comportamento.

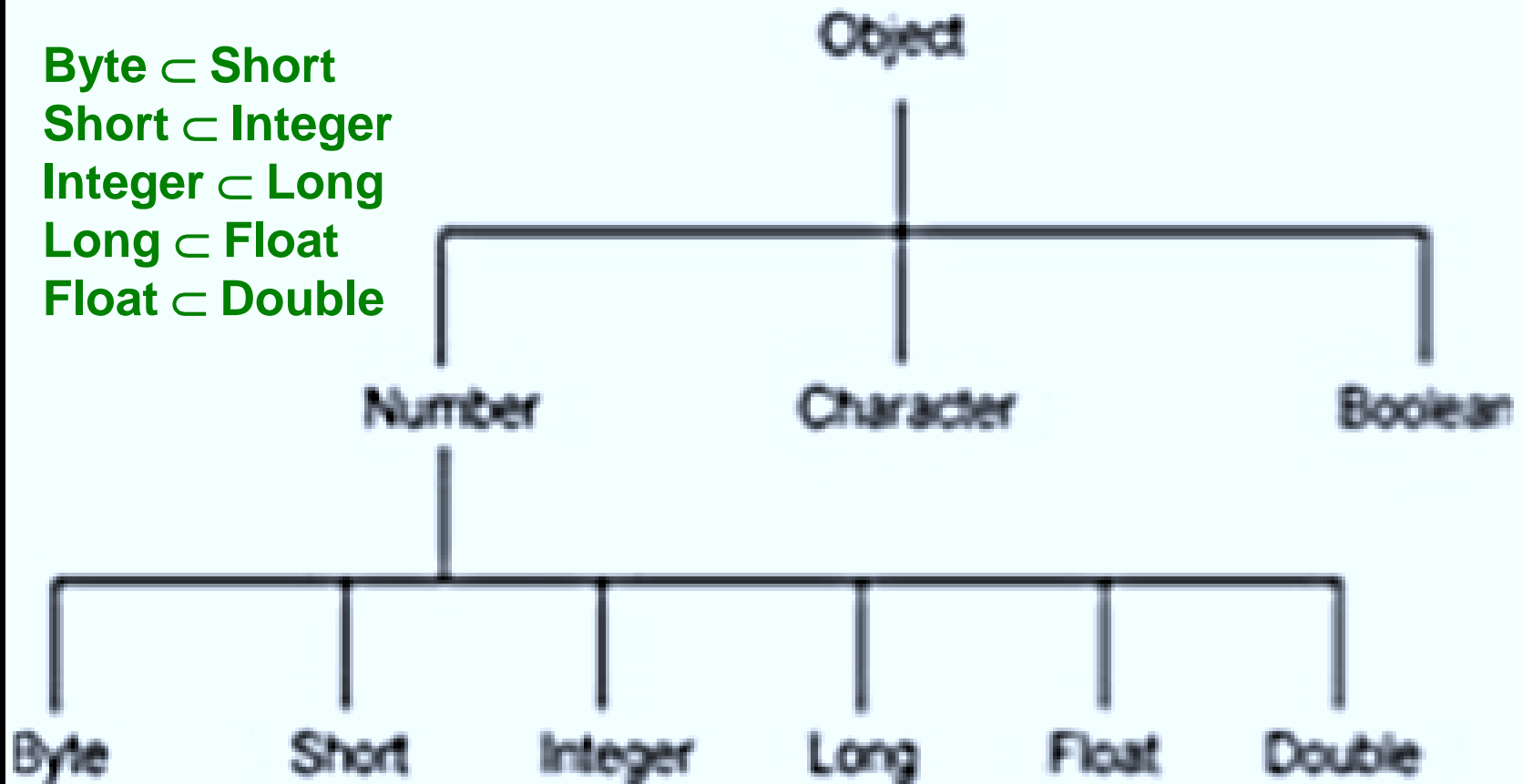


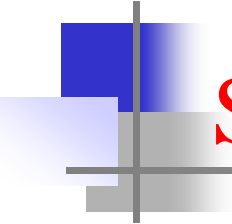
# Princípio da Substituição aplicado a Objetos (Liskov)

- ❑ Sem aplicar o princípio de Liskov a hierarquia de classes poderia gerar graves confusões lógicas, por exemplo, um método cuja a funcionalidade seria somar dois números, poderia se criar um método polimórfico em uma subclasse com o mesmo nome soma com parâmetros diferentes mas cuja a funcionalidade seria um outra diferente, como calcular o imposto de renda, o nome do método polimórfico seria o mesmo, mas uma enganação lógica com um comportamento bem diferente do original o que fere o princípio de Liskov e as boas práticas da POO.

# Exemplo do uso do Princípio da Substituição de Liskov aplicado a Objetos do tipo numérico em Java

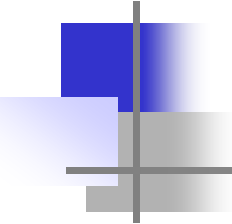
Byte  $\subset$  Short  
Short  $\subset$  Integer  
Integer  $\subset$  Long  
Long  $\subset$  Float  
Float  $\subset$  Double





# Exemplo do uso do Princípio da Substituição de Liskov aplicado a Objetos do Tipo numérico

- ❑ Neste exemplo, um objeto S seja um objeto do tipo número inteiro (Integer) e T um objeto do tipo número real (Float) então pode-se um objeto do tipo T (número real) substituir um objeto do tipo S (número inteiro).
  - Em uma notação da Teoria dos Conjuntos temos que **Float**  $\supset$  **Integer** (Float contém Integer) ou, alternativamente pode-se escrever **Integer**  $\subset$  **Float** (Integer está contido em Float).

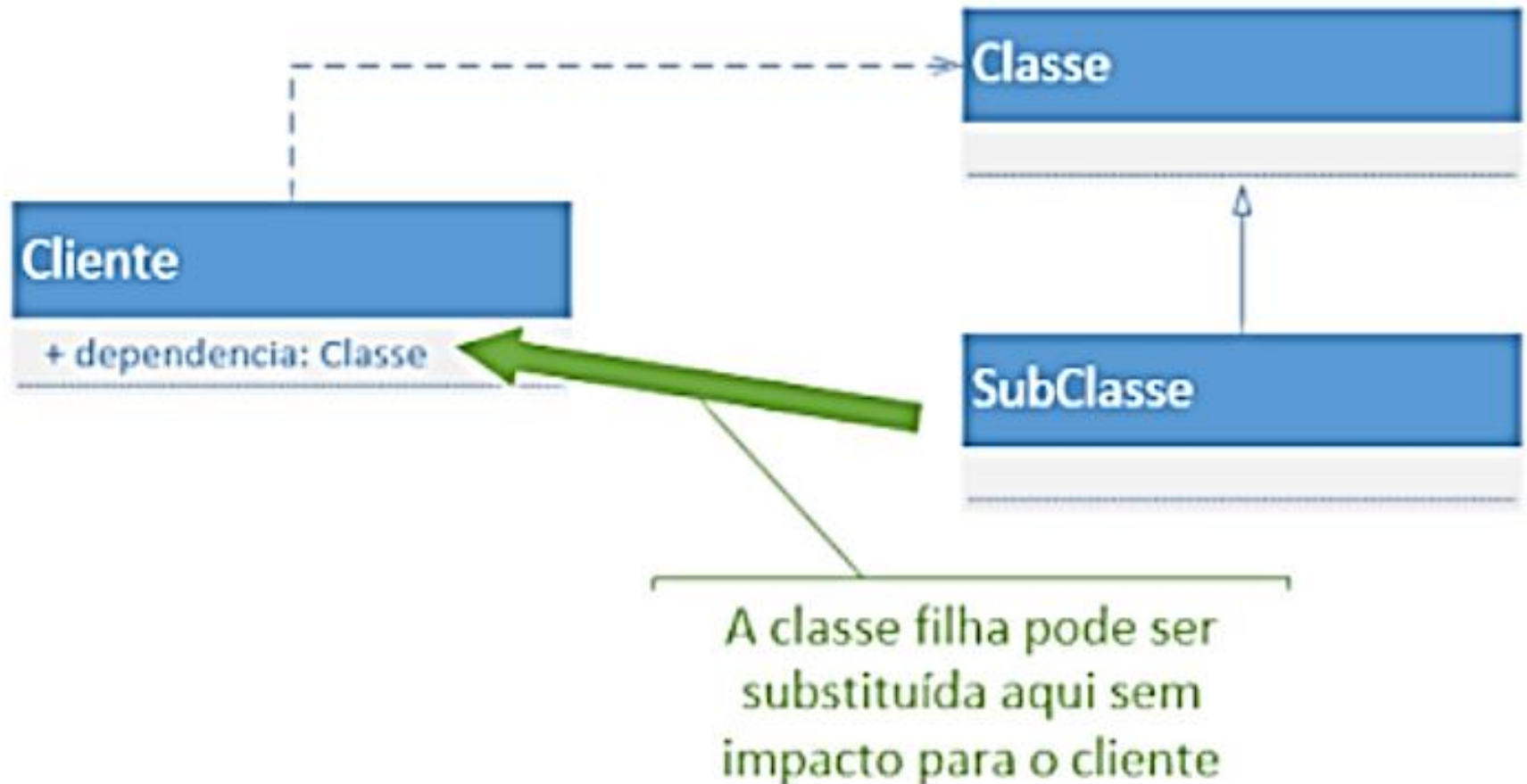


# Exemplo do uso do Princípio da Substituição de Liskov aplicado a Objeto envolvendo herança de classes

- ❑ Neste outro exemplo, vamos aplicar este princípio de forma mais ampla, diz que:
  - “Se  $q(x)$  é uma propriedade demonstrável dos objetos  $x$  de tipo  $T$ . Então  $q(y)$  deve ser verdadeiro para objetos  $y$  de tipo  $S$  onde  $S$  é um subtipo de  $T$ .”
  - Na prática atesta que eu posso substituir uma instância de uma classe por outra instância que seja de uma subclasse da primeira, sem que isso altere o comportamento do sistema.



# Exemplo do uso do Princípio da Substituição aplicado a Objeto em um Diagrama de Classe envolvendo herança





# Ensino da Linguagem Java

Uso da palavra-chave final e  
suas implicações no  
Polimorfismo



# Operação polimórfica em Java

---

- ❑ Ocorre que todas as operações em Java podem ser polimórficas, exceto as qualificadas com a palavra chave **final**.
- ❑ Observação importante:
  - Em Java uma subclasse não é obrigada a sobrepor versões especializadas dos métodos da sua superclasse.



# Palavra chave Final em Java

- ❑ Variáveis declaradas como **final** indicam:
  - que elas não podem ser modificadas depois de declaradas;
  - que devem ser inicializadas quando declaradas e
  - que essas variáveis representam valores constantes.



# Palavra chave Final em Java

❑ Pode-se declarar métodos com o modificador **final**.

- Um método declarado final em uma superclasse não pode ser sobrescrito em uma subclasse.
- Os métodos declarados private são implicitamente final, porque é impossível sobrescrevê-los em uma subclasse (embora a subclasse possa declarar um novo método com a mesma assinatura do método private na superclasse).



# Palavra chave Final em Java

- ❑ Pode-se declarar classes com o modificador **final**.
  - Uma classe declarada com final não pode ser uma superclasse (isto é uma classe não pode estender uma classe final).
  - Todos os métodos em uma classe final são implicitamente final.
  - Tornar uma classe final impede que programadores criem subclasses que poderia driblar as restrições de segurança.



# Palavra chave Final em Java

- ❑ Pode-se declarar classes com o modificador **final**.
  - A classe String é um exemplo de classe final.
  - Esta classe não pode ser estendida, portanto programas que usem a classe String podem contar com a funcionalidade dos objetos String conforme especificado na API Java.



# Recursos da palavra-chave **final** em Java

- ❑ Deve-se inicializar as variáveis finais durante a declaração, caso contrário, podemos inicializar apenas no construtor.
- ❑ Java não suporta a palavra-chave **final** para construtores.
- ❑ Os valores finais das variáveis não podem ser alterados.
- ❑ Não podemos herdar de uma classe **final**.
- ❑ Java não permite sobrescrever um método **final**.
- ❑ Por padrão, todas as variáveis dentro de uma interface são **final**.





---

# Ensino da Linguagem Java

Exemplos de Polimorfismo  
com Herança em Java



# *Exemplo*

## *Classe Super e Sub*

```
class Super
```

```
{  
    public void print()  
    {  
        System.out.printf("Método print da superclasse\n");  
    }  
}
```

```
class Sub extends Super
```

```
{  
    //sobrescreve o metodo da superclasse  
    public void print()  
    {  
        System.out.printf(" Método print da subclasse\n");  
    }  
}
```



# *Exemplo*

## *Classe Testa*

```
public class Testa
{
    public static void main(String args[])
    {
        //associa uma referencia da superclasse a uma variavel da superclasse
        Super sup = new Super();
        //associa uma referencia da subclasse a uma variavel da subclasse
        Sub sub = new Sub();
        //associa uma referencia da subclasse a uma variavel da superclasse
        Super poli = new Sub();

        //invoca o metodo da superclasse usando uma variavel da superclasse
        sup.print();
        //invoca o metodo da subclasse usando uma variavel da subclasse
        sub.print();
        //invoca o metodo da subclasse usando uma variavel da superclasse
        poli.print();
    }
}
```

# Executando no BlueJ

The screenshot displays the BlueJ IDE interface. The main workspace shows a class hierarchy with a class named **Super** and a subclass named **Testa**, connected by a dashed line with an arrow pointing from **Testa** to **Super**. On the left sidebar, there are buttons for **Nova Classe...**, a right-pointing arrow, and **Compilar**. Below these are sections for **Teamwork** (with a **Share...** button) and **Testing** (with an **Executar Testes** button and a **gravando** status indicator). A terminal window titled "BlueJ: BlueJ: Janela de Terminal - Polimorfismo" is open in the foreground, showing the following output:

```
Opções
Método print da superclasse
Método print da subclasse
Método print da subclasse
```

At the bottom of the terminal window, a message reads: "Can only enter input while your programming is r".



# Explicação do Exemplo

- ❑ Quando uma variável da superclasse contém uma referência a um objeto da subclasse, e esta referência é utilizada para invocar um método, a versão da subclasse é utilizada
  - O compilador Java permite isto por causa da relação de herança;
  - Um objeto da subclasse é um objeto da superclasse
    - O contrário não é verdadeiro.
  - Quando o compilador encontra uma chamada a um método através de uma variável, é verificado se o método pode ser chamado, de acordo com a classe da variável;



# Explicação do Exemplo

---

## □ Continuação:

- Se a classe contiver (ou herdar) uma declaração do método, a chamada é compilada;
- Em tempo de execução, a classe do objeto referido pela variável determina qual método será utilizado.

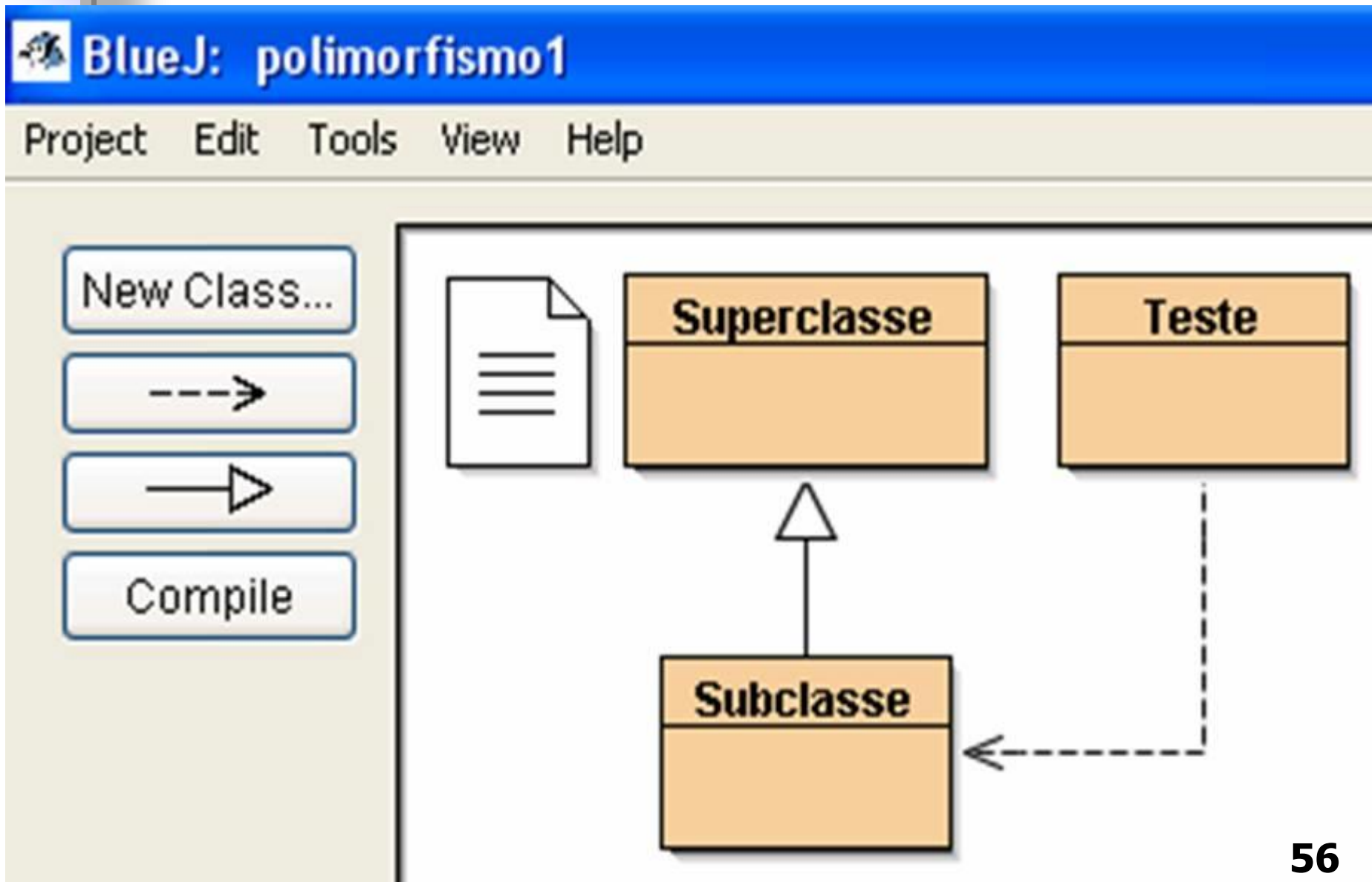


# Outro Exemplo em Java de polimorfismo

---

- ❑ Neste exemplo será abordado a invocação de um método polimórfico com a mesma assinatura que tem duas implementações diferentes:
  - uma na superclasse e
  - outra na subclasse.
- ❑ O exemplo também aborda o uso das palavras chaves **this** e **super** discutida em aula anteriores.

# Exemplo no BlueJ







# Código Fonte

---

```
public class Superclasse {  
    public String nome( ) {  
        // Método nome (polifórmico superclasse)  
        return "Superclasse";  
    } // fim do método polimórfico nome  
} // fim da classe Superclasse
```



# Código Fonte

---

```
public class Subclasse extends  
    Superclasse {  
    public String nome( ) {  
        // Método nome (polifórmico subclasse)  
        return "Subclasse";  
    } // fim do método polimórfico nome
```



# Código Fonte

---

```
public void mostra( ) {  
    Superclasse superclasse = (Superclasse)this;  
    // Observe o uso da palavra chave this e super  
    System.out.println("this.nome( ) = " + this.nome());  
    System.out.println("superclasse.nome( ) = " +  
                        superclasse.nome());  
    System.out.println("super.nome() = " +  
                        super.nome( ));  
} // fim do método mostra
```



# Código Fonte

---

```
public void mostra01( ) {  
    Superclasse superclasse = new Superclasse ( );  
    System.out.println("this.nome( ) = " + this.nome( ));  
    System.out.println("superclasse.nome( ) = " +  
                        superclasse.nome( ));  
    System.out.println("super.nome( ) = " +  
                        super.nome( ));  
    } // fim do método mostra01  
} // fim da classe Subclasse
```



# Código Fonte

---

```
public class Teste {  
    public static void main(String args [ ]) {  
        Subclasse teste = new Subclasse();  
        System.out.println("\nInvocando o método mostra\n");  
        teste.mostra();  
        System.out.println("Invocando o método mostra01\n");  
        teste.mostra01();  
    } // fim do método main  
} // fim da classe Teste
```

# Executando no BlueJ

The screenshot shows the BlueJ IDE interface. The main window, titled "BlueJ: polimorfismo1", contains a class diagram with three classes: "Superclasse", "Subclasse", and "Teste". "Subclasse" is a subclass of "Superclasse", indicated by a solid line with an open triangle arrow. "Teste" is a test class, indicated by a diagonal line in its bottom-right corner. A dashed arrow points from "Teste" to "Subclasse". On the left side of the main window, there are four buttons: "New Class...", a dashed arrow button, a solid arrow button, and "Compile". Below the main window, there is a "Terminal Window - polimorfismo1" pane. It has an "Options" tab and contains the following text:

```
Invocando o método mostra

this.nome() = Subclasse
superclasse.nome() = Subclasse
super.nome() = Superclasse

Invocando o método mostra01

this.nome() = Subclasse
superclasse.nome() = Superclasse
super.nome() = Superclasse
```